Variable Precision Floating Point Division and Square Root

Miriam Leeser, Xiaojun Wang
Department of Electrical and Computer Engineering
Northeastern University, Boston, MA 02115
mel,xjwang@ece.neu.edu

Division and square root are important operations in many high performance signal processing applications including matrix inversion, vector normalization, least squares lattice filters and Cholesky decomposition. We have implemented floating point division and square root designs for our VHDL variable precision floating point library. These designs are implemented in VHDL and are designed to make efficient use of FPGA hardware.

Both the division [1] and square root [2] algorithms are based on table lookup and Taylor series expansion. These algorithms are particularly well-suited for implementation on an FPGA with embedded RAM and embedded multipliers such as the Altera Stratic and Xilinx Virtex2 devices. The division and square root components have been incorporated into the framework of our variable precision floating-point library.

1 Variable Precision Floating-Point Library

Our parameterized floating-point library is composed of three parts: format control, arithmetic operations, and format conversion. Format control includes modules `denorm` and `rnd_norm`. The first is used for denormalizing (introduction of the implied one bit) and the second is used for rounding and normalizing. Format conversion includes modules `fix2float` and `float2fix`. The first is used for converting from fixed-point representation (both unsigned and signed) to floating-point representation and the second converts in the other direction. Arithmetic operations include modules `fp_add`, `fp_sub` and `fp_mul` for floating-point addition, subtraction and multiplication respectively. We recently added floating-point division (`fp_div`) and floating-point square root (`fp_sqrt`). For both floating-point division and square root, we use the small table-lookup method with small multipliers [1, 2]. These algorithms are both small and elegant. Our result shows that these algorithms are very well suited to FPGA implementations, and lead to a good tradeoff of area and latency. Some features of our library are:

- Our parameterized floating-point library is a superset of all the previously published floating-point formats including IEEE standard format.
- Our library is flexible. It supports the creation of custom format floating-point datapaths, as well as hybrid fixed and floating-point implementations.
- Our library is more complete than all other earlier work with a separate normalization unit, rounding with support for both "round to zero" and "round to nearest", and some error handling features.
- Each component in our library has synchronization signals to aid in the creation of pipelines.

2 Division and Square Root

The division and square root we built are based on previously published algorithms [1, 2]. Both of these algorithms are based on Taylor Series and use both small table-lookups and small multipliers to obtain the first few terms of the Taylor Series. These algorithms are both simple and elegant, and very well suited to FPGA implementations. They are also non-iterative algorithms, unlike other implementations of division and square root based on Newton-Raphson. This allows these components to be easily integrated into a larger pipelined design built with other library modules without decreasing the throughput of the whole design.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01 FEB 2005	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Variable Precision Floating Point Division and Square Root	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Department of Electrical and Computer Engineering Northeastern University, Boston, MA 02115**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**36**	

Table 1: Cost and Performance for Floating-Point Division

Floating Point Format	8(2,5)	16(4,11)	24(6,17)	32(8,23)
number of slices	69 (1%)	110 (1%)	254 (1%)	335 (2%)
number of BlockRAM	1 (1%)	1 (1%)	1 (1%)	7 (7%)
number of 18x18 embedded multiplier	2 (2%)	2 (2%)	8 (8%)	8 (8%)
clock period (ns)	8	10	9	9
maximum frequency (MHz)	124	96	108	110
number of clock cycles to generate final results	10	10	14	14
latency(ns) = clock × number of clock cycles	80	105	129	127
throughput (million results per second)	124	96	108	110

Table 1 shows the cost and performance of four different floating-point formats (including IEEE single precision format) for division. Results for square root are similar. All our designs are specified in VHDL and mapped to Xilinx Virtex-II XC2v3000-4 FPGA. All area and timing results in the above tables are those reported by the Xilinx tools. Our results show that both the area and the latency of our floating-point division and square root implementations are small. For IEEE single precision format division, it takes 14 clock cycles to generate final results with a 9ns clock period, so the latency is only 127ns. Since it can be fully pipelined, the throughput is high at 110 million results per second. This design takes only 2% of the slices, 7% of the BlockRAMs, and 8% of the 18x18 embedded multipliers on the FPGA chip, which is a very small design. Our floating-point square root shows the similar good tradeoff of area, latency and throughput.

To demonstrate the division implementation, we are incorporating it into our implementation of the K-means clustering algorithm applied to multispectral satellite images [3] K-means clustering is an iterative algorithm where the total number of clusters is known in advance. The algorithm works as follows. First means are initialized using a hierarchical method. During each iteration, each pixel of the image is assigned to the closest cluster based on the distance between each pixel and each of the K cluster centers. At the end of one iteration, the new mean of each cluster is calculated based on the new pixel assignments and is used for the next iteration as the center of each cluster. To obtain the new mean of each cluster, an accumulator and a counter are associated with each cluster. Once a pixel is assigned to a cluster, the value of the pixel is added to the accumulator and the counter is incremented. The new mean is obtained by dividing the accumulator value by the counter value. In our previous design [3] this mean updating step is done on the host because it requires floating-point division. With our new `fp_div` module, we are able to implement the mean updating in FPGA hardware. This greatly reduces the communication between host and FPGA board and further accelerates the runtime.

References

[1] P. Hung, H. Fahmy, O. Mencer, and M. J. Flynn, "Fast division algorithm with a small lookup table," in *Asilomar Conference on Signals,Systems and Computers*, vol. 2, pp. 1465–1468, November 1999.

[2] M. D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," *IEEE Transactions on Computers*, vol. 49, pp. 628–637, July 2000.

[3] P. Belanovic and M. Leeser, "A library of parameterized modules for floating-point arithmetic and their use," in *High Performance Embedded Computing*, September 2002.

Variable Precision Floating Point Division and Square Root

Albert Conti
Xiaojun Wang
Dr. Miriam Leeser
Rapid Prototyping Laboratory
Northeastern University, Boston MA
http://www.ece.neu.edu/groups/rpl/

Outline

- Project overview
- Library hardware modules
- Floating point divider and square root
- K-means clustering application for multispectral satellite images using the floating point library
- Conclusions and future work

Variable Precision Floating Point Library

- A library of fully pipelined and parameterized floating point modules
- Implementations well suited for state of the art FPGAs
  - Xilinx Virtex II FPGAs and Altera Stratix devices
  - Embedded Multipliers and Block RAM
- Signal/image processing algorithms accelerated using this library

Why Floating Point (FP) ?

Fixed Point

- Limited range
- Number of bits grows for more accurate results
- Easy to implement in hardware

Floating Point

- Dynamic range
- Accurate results
- More complex and higher cost to implement in hardware

Floating Point Representation

Sign	Biased exponent	Mantissa m=1.f (the 1 is hidden)
+/-	e+bias	f

32-bits:	8 bits, bias=127	23+1 bits, IEEE single-precision format
64-bits:	11 bits, bias=1023	52+1 bits, IEEE double-precision format

$$(-1)^{s} * 1.f * 2^{e-BIAS}$$

Why Parameterized FP ?

- Minimize the bitwidth of each signal in the datapath
  - Make more parallel implementations possible
  - Reduce the power dissipation
- Further acceleration
  - Custom datapaths built in reconfigurable hardware using either fixed-point or floating point arithmetic
  - Hybrid representations supported through fixed-to-float and float-to-fixed conversions

Outline

- Project overview
- Library hardware modules
- Floating point divider and square root
- K-means clustering application for multispectral satellite images using the floating point library
- Conclusions and future work

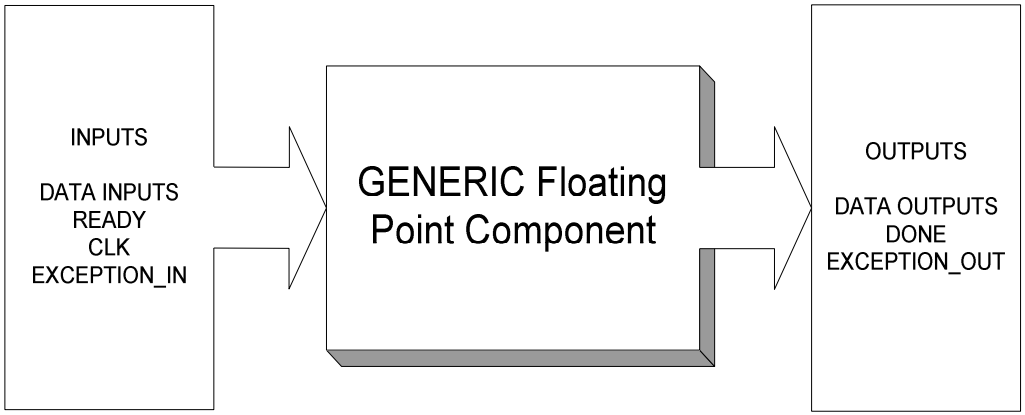
Parameterized FP Modules

- Arithmetic operation
  - **fp_add** : floating point addition
  - **fp_sub** : floating point subtraction
  - **fp_mul** : floating point multiplication
  - **fp_div** : floating point division
  - **fp_sqrt** : floating point square root
- Format control
  - **denorm** : introducing implied integer digit
  - **rnd_norm** : rounding and normalizing
- Format conversion
  - **fix2float** : converting from fixed point to floating point
  - **float2fix** : converting from floating point to fixed point

What Makes Our Library Unique ?

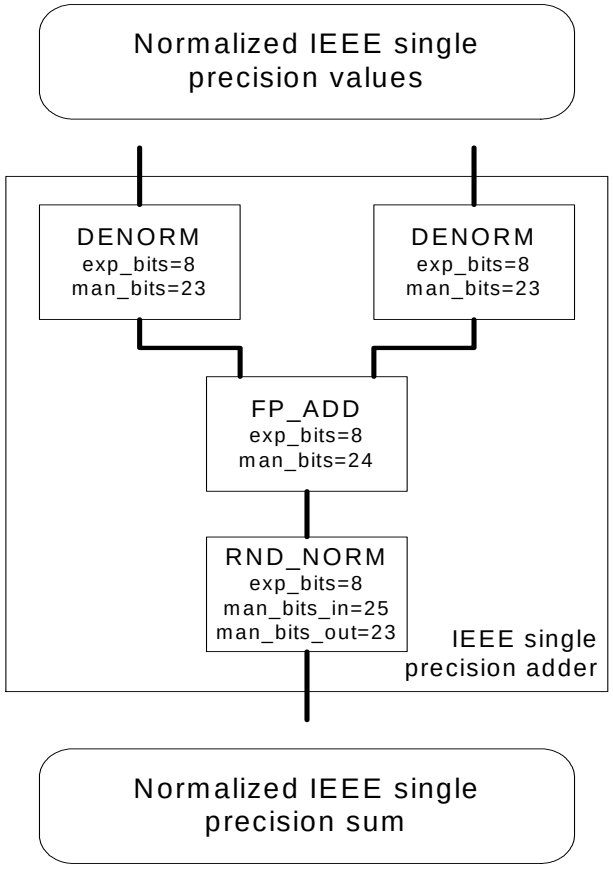
- A superset of all floating point formats
  - including IEEE standard format
- Parameterized for variable precision arithmetic
  - Support custom floating point datapaths
  - Support hybrid fixed and floating point implementations
- Support fully pipelining
  - Synchronization signals
- Complete
  - Separate normalization
  - Rounding (“round to zero” and “round to nearest”)
  - Some error handling

Generic Library Component



- Synchronization signals for pipelining
  - READY and DONE
- Some error handling features
  - EXCEPTION_IN and EXCEPTION_OUT

One Example
- Assembly of Modules



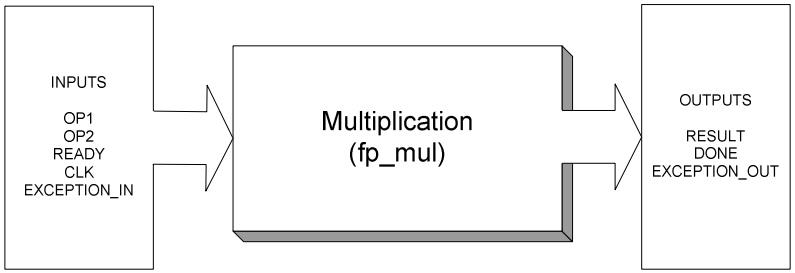
$2 \times \text{denorm}$

$+ 1 \times \text{fp\_add}$

$+ 1 \times \text{rnd\_norm}$

$= 1 \times$ IEEE single precision adder

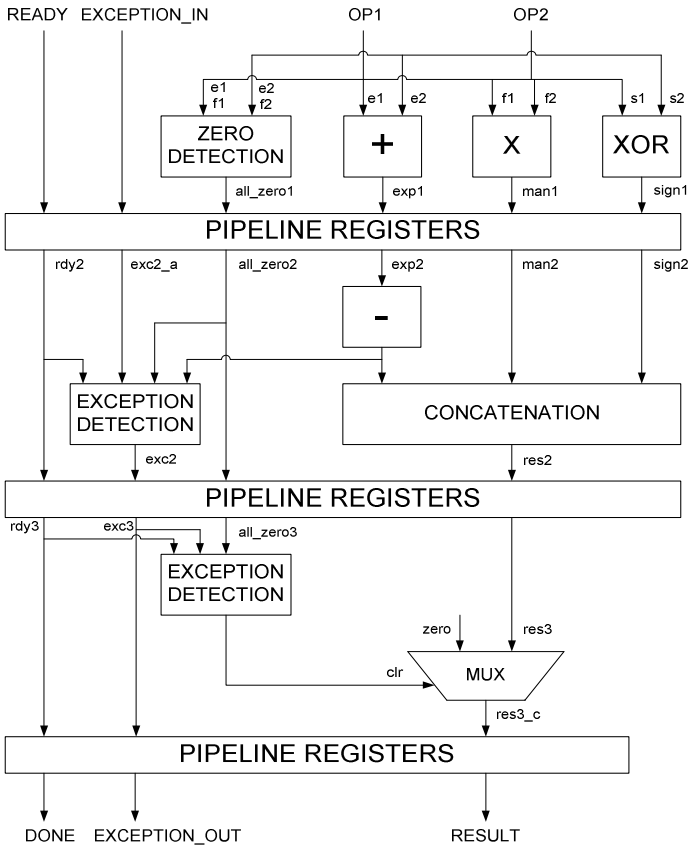
Another Example
- Floating Point Multiplier



$$(-1)^{s1} * 1.f1 * 2^{e1-BIAS}$$

$$x \quad (-1)^{s2} * 1.f2 * 2^{e2-BIAS}$$

$$(-1)^{s1 \text{ xor } s2} * (1.f1*1.f2) * 2^{(e1+e2-BIAS)-BIAS}$$



Latency

Module	Latency (clock cycles)
denorm	0
rnd_norm	2
fp_add / fp_sub	4
fp_mul	3
fp_div	14
fp_sqrt	14
fix2float(unsigned/signed)	4/5
float2fix(unsigned/signed)	4/5

Clock rate of each module is similar

Outline

- Project overview
- Library hardware modules
- Floating point divider and square root
- K-means clustering application for multispectral satellite images using the library
- Conclusions and future work

Algorithms for Division and Square Root

- Division
  - P. Hung, H. Fahmy, O. Mencer, and M. J. Flynn, “Fast division algorithm with a small lookup table," *Asilomar Conference*,1999
- Square Root
  - M. D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand, “Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," *IEEE Transactions on Computers,* vol. 2, pp. 628-637, 2000

Why Choose These Algorithms?

- Both algorithms are simple and elegant
  - Based on Taylor series
  - Use small table-lookup method with small multipliers
- Very well suited to FPGA implementations
  - BlockRAM, distributed memory, embedded multiplier
  - Lead to a good tradeoff of area and latency
- Can be fully pipelined
  - Clock speed similar to all other components

Division Algorithm

Dividend X and divisor Y are 2m-bit fixed-point number $\in$ [1,2)

$$X = 1 + 2^{-1} x_1 + 2^{-2} x_2 + \ldots + 2^{-(2m-1)} x_{2m-1}$$

$$Y = 1 + 2^{-1} y_1 + 2^{-2} y_2 + \ldots + 2^{-(2m-1)} y_{2m-1}$$

,where $x_i, y_i \in \{0,1\}$

Y is decomposed into higher order bit part Y_h and lower order bit part Y_l , which are defined as

$$Y_h = 1 + 2^{-1} y_1 + 2^{-2} y_2 + \ldots + 2^{-m} y_m$$

$$Y_l = 2^{-(m+1)} y_{m+1} + \ldots + 2^{-(2m-1)} y_{2m-1}$$

,where $Y_h > 2^m \bullet Y_l$

Division Algorithm – Continue

Using Taylor series

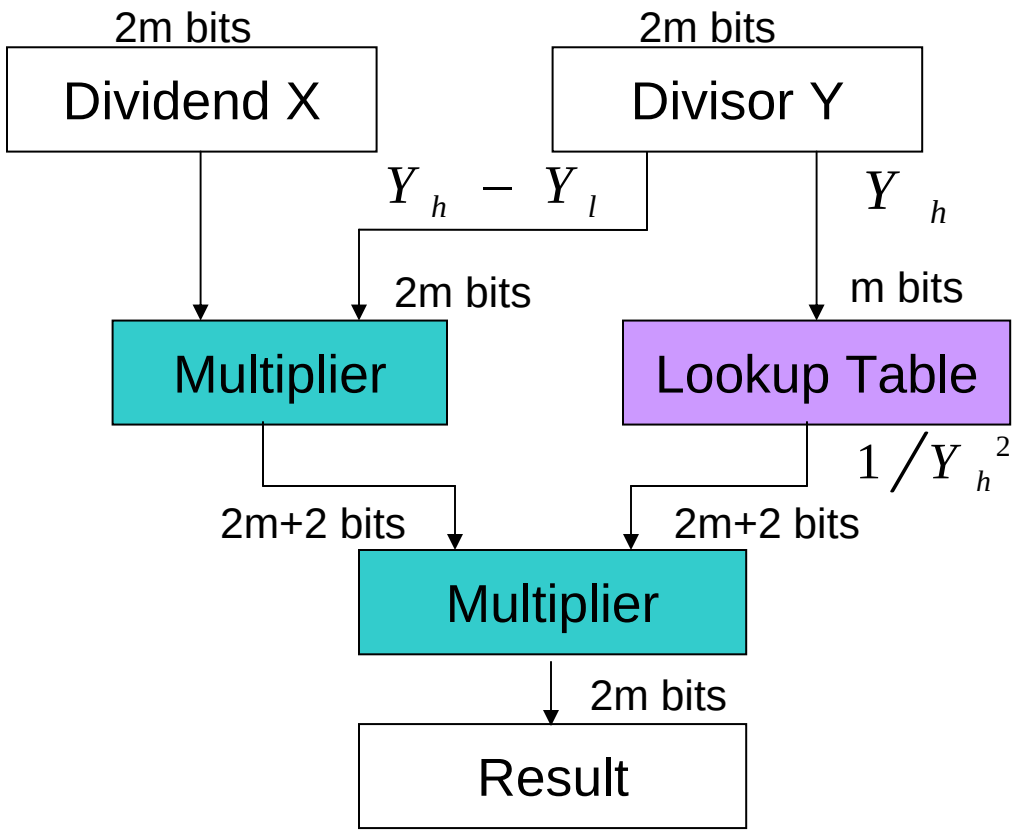
$$\frac{X}{Y} = \frac{X}{Y_h + Y_l} = \frac{X}{Y_h}\left(1 - \frac{Y_l}{Y_h} + \frac{Y_l^2}{Y_h^2} - \ldots\right)$$

$$\approx X \times (Y_h - Y_l) \times \frac{1}{Y_h^2}$$

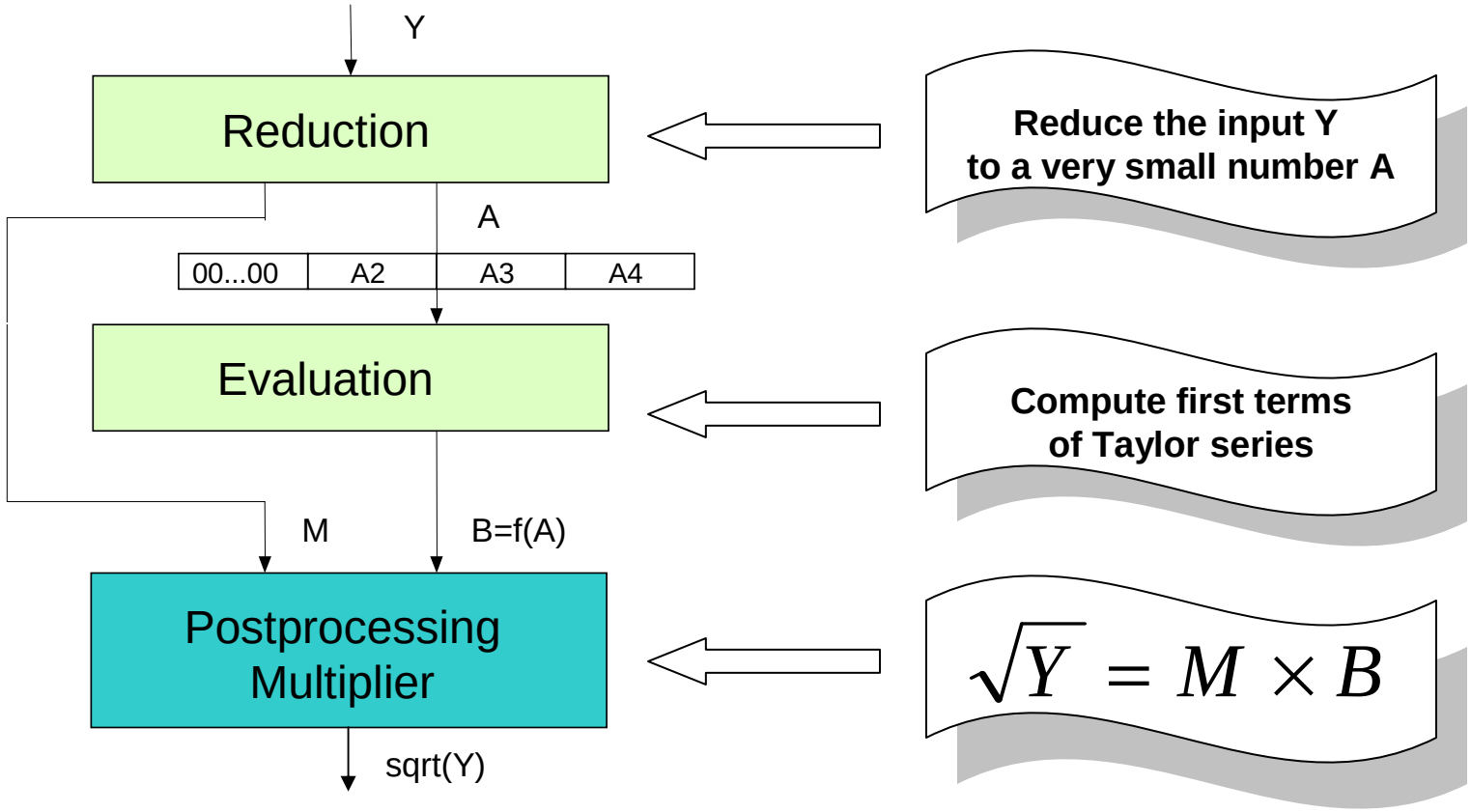
Error less than ½ ulp

Two multipliers and one Table-Lookup are required

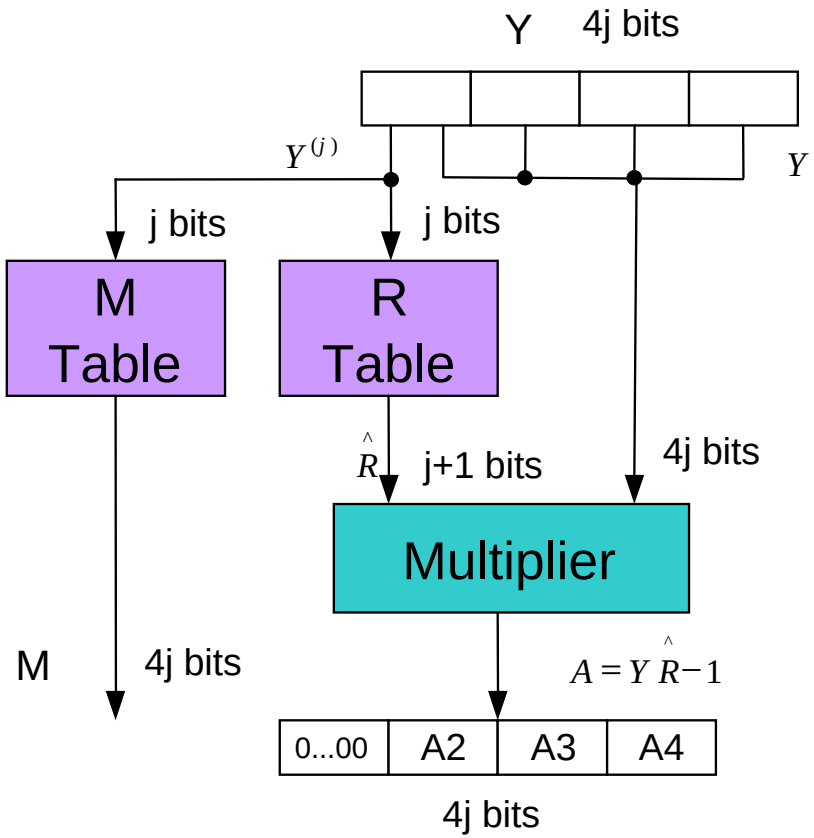
Division – Data Flow



Square Root – Data Flow



Square Root – Reduction

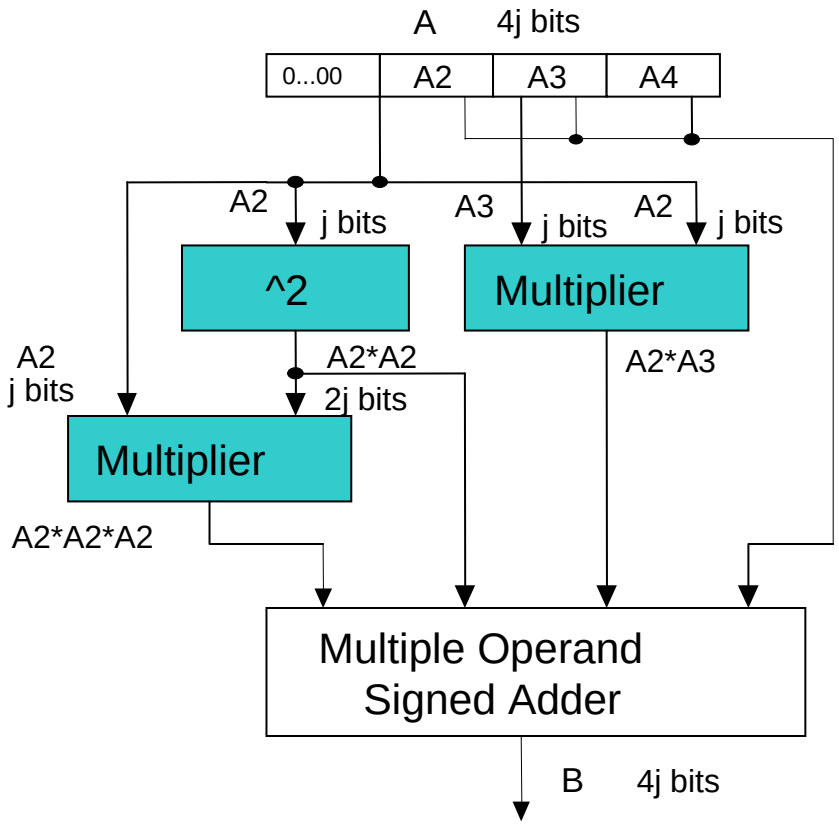


$$\hat{R} = 1 / Y^{(j)}$$

$$M = 1 / \sqrt{\hat{R}}$$

$$A = Y \times R - 1$$

Square Root - Evaluation

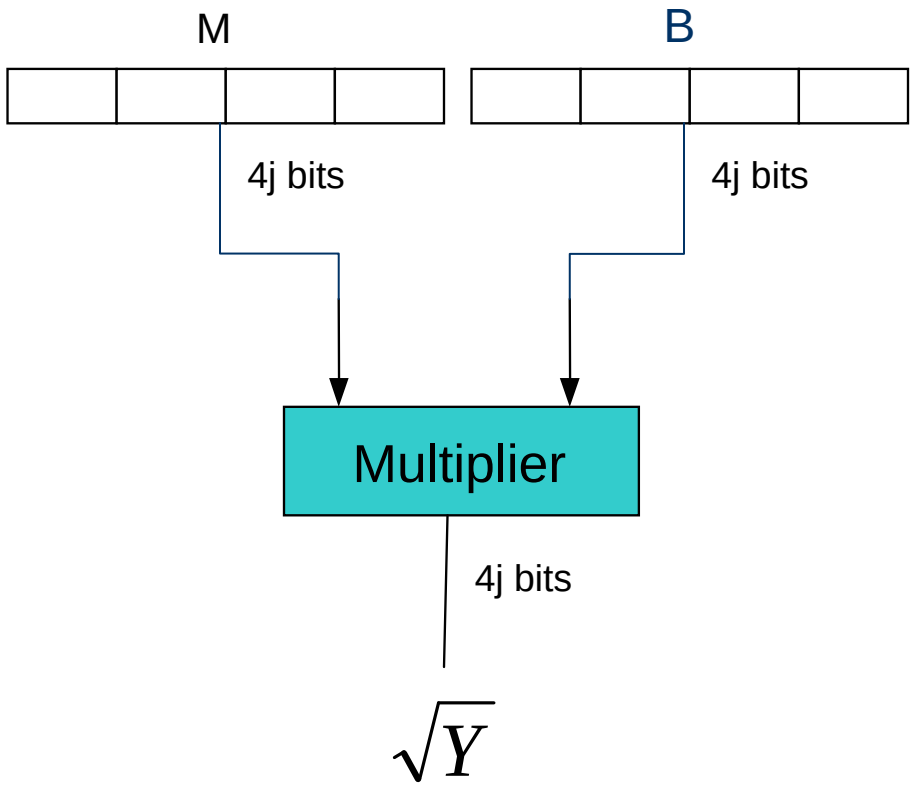


$$A = A_2 z^2 + A_3 z^3 + A_4 z^4 \ldots$$

$$B = \sqrt{1+A}$$

$$= 1 + \frac{2}{A} - \frac{1}{8} A_2^2 z^4 - \frac{1}{4} A_2 A_3 z^5 + \frac{1}{16} A_2^3 z^6$$

Square Root – Post Processing



$$\sqrt{Y} = M \times B$$

Results Mapping to Hardware

- Designs specified in VHDL
- Mapped to Xilinx Virtex II FPGA (XC2V3000)
  - System clock rates up to 300 MHz
  - Density up to 8M system gates
  - 14,336 slices
  - 96 18x18 Embedded Multipliers
  - 96 18Kb BlockRAM (1,728 Kb)
  - 448 Kb Distributed Memory
- Currently targeting Annapolis Wildcard-II

Results - FP Divider on a XC2V3000

Floating Point Format	8(2,5)	16(4,11)	24(6,17)	32(8,23)
# of slices	69 (1%)	110 (1%)	254 (1%)	335 (2%)
# of BlockRAM	1 (1%)	1 (1%)	1 (1%)	7 (7%)
# of 18x18 Embedded Multiplier	2 (2%)	2 (2%)	8 (8%)	8 (8%)
Clock period (ns)	8	10	9	9
Maximum frequency (MHz)	124	96	108	110
# of clock cycles to obtain final results	10	10	14	14
Latency (ns)=clock period x # of clock cycles	80	105	129	127
Throughput (million results/second)	124	96	108	110

The last column is the IEEE single precision floating point format

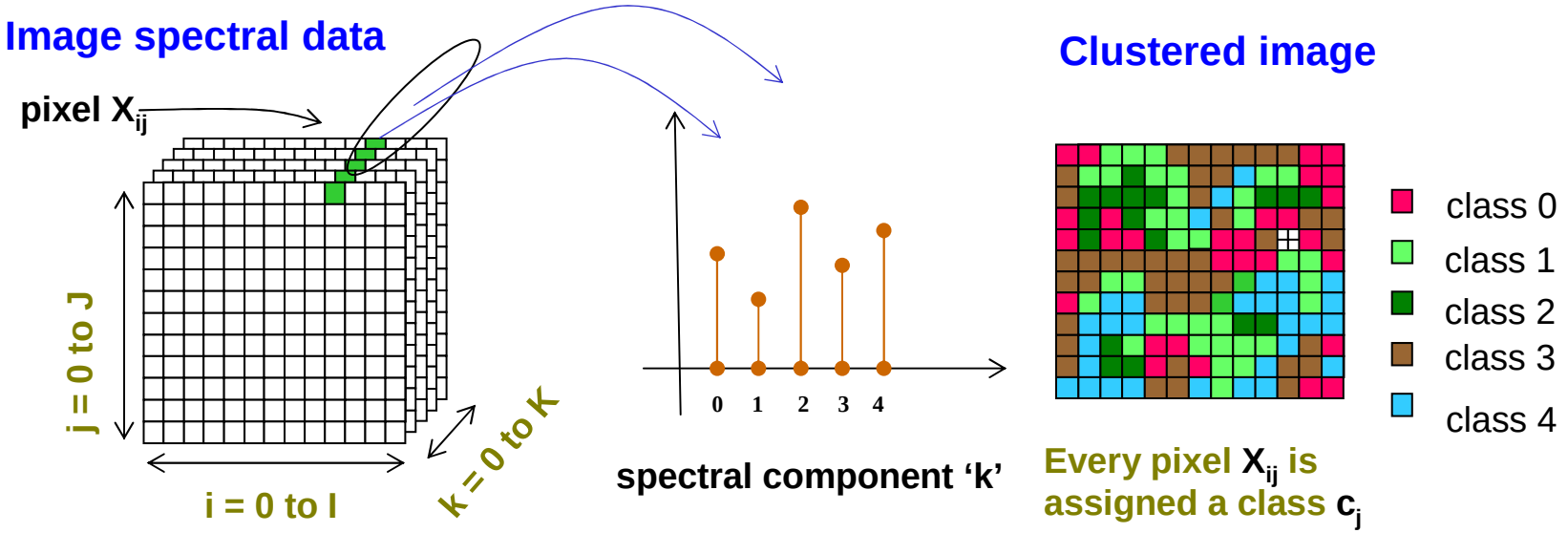
Results - FP Square Root on a XC2V3000

Floating Point Format	8(2,5)	16(4,11)	24(6,17)	32(8,23)
# of slices	113 (1%)	253 (1%)	338 (2%)	401 (2%)
# of BlockRAM	3 (3%)	3 (3%)	3 (3%)	3 (3%)
# of 18x18 Embedded Multiplier	4 (4%)	5 (5%)	9 (9%)	9 (9%)
Clock period (ns)	10	9	11	12
Maximum frequency (MHz)	103	112	94	86
# of clock cycles to obtain final results	9	12	13	13
Latency (ns)=clock period x # of clock cycles	88	107	138	152
Throughput (million results/second)	103	112	94	86

Outline

- Project overview
- Library hardware modules
- Floating point divider and square root
- K-means clustering application for multi-spectral satellite images using the library
- Conclusions and future work

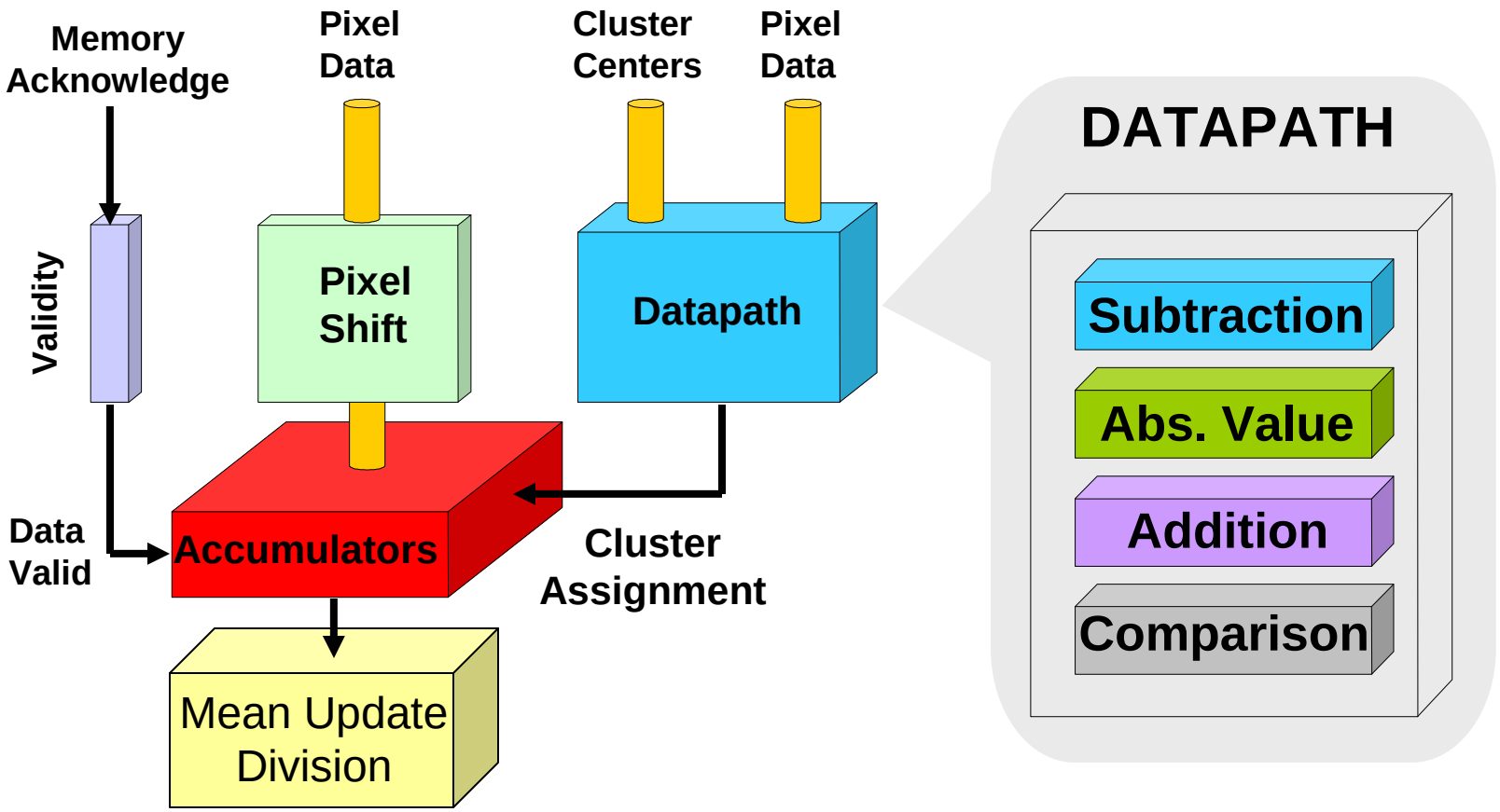
Application : K-means Clustering for Multispectral Satellite Images



K-means – Iterative Algorithm

- Each cluster has a center (mean value)
  - Initialized on host
  - Initialization done once for complete image processing
- Cluster assignment
  - Distance (Manhattan norm) of each pixel and cluster center
- Accumulation of pixel value of each cluster
- Mean update via dividing the accumulator value by number of pixels
- **Division step now executed on-chip with fp_divide to improve performance**

K-means Clustering – Functional Units



Outline

- Project overview
- Library hardware modules
- Floating point divider and square root
- K-means clustering application for multispectral satellite images using the library
- Conclusions and future work

Conclusion

- A Library of fully pipelined and parameterized hardware modules for floating point arithmetic
- Flexibility in forming custom floating point formats
- New module fp_div and fp_sqrt have small area and low latency, are easily pipelined
- K-means clustering algorithm applied to multispectral satellite images makes use of fp_div

Future Work

- More applications using
  - fp_div and fp_sqrt
- New library modules
  - ACC, MAC, INV_SQRT
- Use floating point lib to implement floating point coprocessor on FPGA with embedded processor

For Additional Information

Rapid Prototyping Laboratory
Northeastern University, Boston MA
http://www.ece.neu.edu/groups/rpl/

aconti , xjwang , mel @ece.neu.edu